นริศ วิชาธีรสกุล

วินโดวส์โปรแกรมมิง
ตอนที่ 4 : คีย์บอร์ดอินพุต

ศึกษากลไกการรับข้อมูล และเทคนิคการเขียนโปรแกรม รายงานการทำงานของคีย์บอร์ด

ในฉบับก่อนหน้านี้มีการเปิดตัววินโดวส์โปรแกรม (Window Program) สำหรับตอนนี้เรามาดูในเรื่องเกี่ยวกับอินพุต (Input) คิดว่าผู้อ่านคงเคยผ่านการเขียนโปรแกรมที่เกี่ยวกับอินพุตมาบ้างแล้ว อย่างน้อยก็รับข้อมูลผ่านทางแป้นพิมพ์ผ่านอินเตอร์รัพต์ (Interrupt 9h หรือ 16h) มากขึ้นหน่อย บางคนก็อาจเคยเขียนเม้าส์อินเตอร์เฟส (Mouse Interface) ซึ่งมีความคล้ายคลึงกับเรื่องที่เราจะพูดถึง

ในส่วนอินพุตที่ผมนำมาเสนอจะแบ่งหัวเรื่องดังนี้

- คีย์บอร์ดอินพุต (Keyboard Input) เป็นส่วนอินพุตที่ได้จากแป้นพิมพ์
- เม้าส์อินพุต (Mouse Input) เป็นส่วนอินพุตที่ได้จากเม้าส์
- ไทม์เมอร์อินพุต (Timer Input) เป็นส่วนอินพุตพิเศษ (ที่ดูไม่น่าเกี่ยวกับอินพุตเท่าไหร่ แต่โดยเนื้อแท้แล้วเราก็คงต้องจัดไว้ในกลุ่มอินพุต)

คีย์บอร์ดอินพุต

เมื่อกดแป้นตัวอักษรแล้วค่าสแกนโค้ดของแป้นพิมพ์จะถูกส่งไปยังส่วนควบคุมบนเมนบอร์ดและเกิดฮาร์ดแวร์อินเตอร์รัพต์ (Hardware Interrupt) 09h ขึ้น ถ้าอยู่ในสิ่งแวดล้อมของดอส (DOS Environment) ผลของอินเตอร์รัพต์จะไปประมวลผลส่วนคีย์บอร์ดรูทีนในรอมไบออสตามแอดเดรสตำแหน่งไบต์ที่ 36-39 (offset : segment) ของตารางอินเตอร์รัพต์

ถ้าอยู่ในวินโดวส์ สิ่งที่เกิดขึ้นจะแตกต่างไปจากสิ่งที่เกิดบนดอส

กลไกการรับข้อมูลทางคีย์บอร์ด

เมื่อรันวินโดวส์ คีย์บอร์ดไดรเวอร์ (Keyboard driver) จะติดตั้งอินเตอร์รัพต์แฮนเดิลเลอร์ (Interrupt Handler) ซึ่งจะทำงานแทนที่คีย์บอร์ดรอมไบออสรูทีนเดิมก่อนที่จะรันวินโดวส์ และจะถูกเรียกทุกครั้งที่มีการกดหรือปล่อยแป้นพิมพ์ โดยที่ค่าสแกนโค้ดจะถูกอินเตอร์รัพต์แฮนเดิลเลอร์อ่านขึ้นมาจาก

พอร์ตจะแปลงไปเป็นค่าสมมติ ซึ่งขอเรียกว่า "รหัสแป้นพิมพ์เทียม" (Virtual Keyboard Value) ซึ่งแสดงค่าและสัญลักษณ์ที่ถูกกำหนดไว้แล้วใน windows.h ดังรูปที่ 1

ไม่เพียงแต่อินเตอร์รัพต์แฮนเดิลเลอร์จะแปลงสแกนโค้ดเป็นเวอร์ชวลคีย์โค้ดแล้วยังเกิดเมสเสจที่สืบเนื่องจากการกดหรือปล่อยใส่ไว้ในเมสเสจคิว (Message Queue) ซึ่งรอให้แอพพลิเคชันนำไปใช้

โฟกัส

สำหรับแป้นพิมพ์เป็นอุปกรณ์ที่ใช้ร่วมกันสำหรับทุกหน้าต่างที่รันอยู่ภายใต้วินโดวส์ ดังนั้นในกรณีที่มีหน้าต่างเปิดอยู่มากกว่าหนึ่งหน้าต่าง เมื่อกดแป้นพิมพ์แล้วจะมีเพียงหน้าต่างเดียวที่มีสถานะที่ได้รับอินพุตไป ซึ่งสถานะนี้เราเรียกว่าโฟกัส (Focus) โดยทั่วไปหน้าต่างที่แอคตีฟอยู่ในขณะนั้นจะได้โฟกัสนี้แต่ก็ไม่จำเป็นเสมอ

คีย์สโตรคเมสเสจ (Keystroke Message)

ในสิ่งแวดล้อมของวินโดวส์ เมื่อเรากดหรือปล่อยแป้นพิมพ์จะเกิดคีย์สโตรคเมสเสจขึ้นในเมสเสจคิว ถ้าในกรณีที่กดแป้นตัวอักษร เมสเสจที่เกิดขึ้นคือ WM_KEYDOWN หรือ WM_SYSKEYDOWN แต่เมื่อปล่อยแป้นตัวอักษร เมสเสจที่เกิดขึ้นจะกลับกันคือ WM_KEYUP หรือ WM_SYSKEYUP

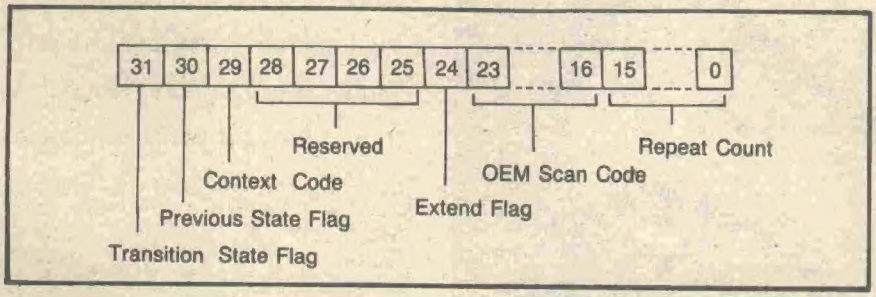
ถ้าดูแล้วคงเกิดข้อสงสัยว่า WM_KEYDOWN และ WM_SYSKEYDOWN หรือ WM_KEYUP และ WM_SYSKEYUP ต่างกันอย่างไร ดูกันง่าย ๆ ว่า WM_SYSKEYDOWN

รูปที่ 1 แสดงรหัสแป้นพิมพ์เทียม

1	1	VK_LBUTTON	
2	2	VK_RBUTTON	
3	3	VK_CANCEL	Ctrl-Break
4	4	VK_MBUTTON	
8	8	VK_BACK	Backspace
9	9	VK_TAB	Tab
C	12	VK_CLEAR	5 or Numeric keypad with Num Lock OFF
D	13	VK_RETURN	Enter
10	16	VK_SHIFT	Shift
11	17	VK_CONTROL	Ctrl
12	18	VK_MENU	Alt
13	19	VK_PAUSE	Pause (or Ctrl-Num Lock)
14	20	VK_CAPITAL	Caps Lock
1B	27	VK_ESCAPE	Esc
20	32	VK_SPACE	Spacebar
21	33	VK_PRIOR	Page Up
22	34	VK_NEXT	Page Down
23	35	VK_END	End
24	36	VK_HOME	Home
25	37	VK_LEFT	
26	38	VK_UP	Up Arrow
27	39	VK_RIGHT	Right Arrow
28	40	VK_DOWN	Down Arrow
29	41	VK_SELECT	< unused >
2A	42	VK_PRINT	< unused >
2B	43	VK_EXECUTE	< unused >
2C	44	VK_SNAPSHOT	Print Screen
2D	45	VK_INSERT	Ins
2E	46	VK_DELETE	Del
2F	47	VK_HELP	< unused >
30-39	48-57	VK_0 to VK_9	0 to 9 above letter keys
41-5A	65-90	VK_A to VK_Z	A to Z
60	96	VK_NUMPAD0	0 on Numeric keypad with Num Lock ON
61	97	VK_NUMPAD1	1 on Numeric keypad with Num Lock ON
62	98	VK_NUMPAD2	2 on Numeric keypad with Num Lock ON
63	99	VK_NUMPAD3	3 on Numeric keypad with Num Lock ON
64	100	VK_NUMPAD4	4 on Numeric keypad with Num Lock ON

รูปที่ 1 (ต่อ)

65	101	VK_NUMPAD5	5 on Numeric keypad with Num Lock ON
66	102	VK_NUMPAD6	6 on Numeric keypad with Num Lock ON
67	103	VK_NUMPAD7	7 on Numeric keypad with Num Lock ON
68	104	VK_NUMPAD8	8 on Numeric keypad with Num Lock ON
69	105	VK_NUMPAD9	9 on Numeric keypad with Num Lock ON
6A	106	VK_MULTIPLY	* on Numeric keypad
6B	107	VK_ADD	+ on Numeric keypad
6C	108	VK_SEPARATOR	< unused >
6D	109	VK_SUBTRACT	- on Numeric keypad
6E	110	VK_DECIMAL	. on Numeric keypad
			on Numeric keypad
6F	111	VK_DEVIDE	/ on Numeric keypad
70	112	VK_F1	F1 function key
71	113	VK_F2	F2 function key
72	114	VK_F3	F3 function key
73	115	VK_F4	F4 function key
74	116	VK_F5	F5 function key
75	117	VK_F6	F6 function key
76	118	VK_F7	F7 function key
77	119	VK_F8	F8 function key
78	120	VK_F9	F9 function key
79	121	VK_F10	F10 function key
7A	122	VK_F11	F11 function key
7B	123	VK_F12	F12 function key
7C	124	VK_F13	F13 function key
7D	125	VK_F14	F14 function key
7E	126	VK_F15	F15 function key
7F	127	VK_F16	F16 function key
90	144	VK_NUMLOCK	Num Lock ?
91	145		Scroll Lock



รูปที่ 2 การเก็บอินฟอร์มเมชันของคีย์สโตรค

และ WM_SYSKEYUP เป็นเมสเสจที่เกิดจากซิสเต็มส์คีย์สโตรค (System Keystroke) ทั้งนี้สืบเนื่องจากการที่กดแป้น Alt ควบคู่กับแป้นอักษรอื่นไปด้วย เช่น Alt-A เป็นต้น

ถ้ากดแป้น Alt ควบคู่ไปกับแป้นใด วินโดวส์จะสงวนเป็นซิสเต็มส์คีย์ (System Key) ซึ่งมักจะใช้ในส่วนของเมนูซิสเต็มส์ (Menu System)

ทั้ง WM_KEYDOWN หรือ WM_SYSKEYDOWN หรือ WM_KEYUP หรือ WM_SYSKEYUP เวลานำไปใช้ในโปรแกรมจะมีพารามิเตอร์ตามมาด้วย 2 พารามิเตอร์ โดยที่ตัวแรกมีขนาด 2 ไบต์ ซึ่งในที่นี้ใช้ชื่อว่า wParam ซึ่งมีค่าเท่ากับ "ค่าของรหัสเทียม" และพารามิเตอร์อีกตัวมีขนาด 4 ไบต์ซึ่งขอใช้ชื่อว่า lParam ซึ่งจะเก็บอินฟอร์เมชันของคีย์สโตรคนั้น ๆ สามารถที่จะแบ่งได้อีก 6 ส่วน ดังรูปที่ 2

รีพีตเคานท์ (Repeat Count) เป็นจำนวนครั้งของคีย์สโตรคที่ซ้ำกันเมื่อมีการกดแป้นพิมพ์แช่ไว้ ทั้งนี้เนื่องจากวินโดวส์จะจัดการเอา WM_KEYDOWN เมสเสจ (WM_SYSKEYDOWN เมสเสจ) ของแป้นที่กดแช่ให้เหลือเพียง 1 เมสเสจ เพื่อป้องกันการโอเวอร์โฟลว์ของเมสเสจคิว

OEM Scan Code คือ คีย์บอร์ดสแกนโค้ดที่สืบเนื่องจากฮาร์ดแวร์สแกนโค้ด ซึ่งโดยปกติแล้วเรามักจะหลีกเลี่ยงไปใช้เวอร์ชวลคีย์โค้ด (Virtual Key Code) แทน เนื่องจากมีความเป็นอิสระจากฮาร์ดแวร์ (Hardware Independence) มากกว่า

Extend Key Flag จะถูกกำหนดเป็น 1 เมื่อคีย์สโตรคที่ได้มาจากแป้นพิมพ์ที่เพิ่มเติมเข้ามาบน IBM Enhanced

Keyboard (คีย์บอร์ดของเครื่อง PC-AT)

Context Code จะถูกกำหนดเป็น 1 เมื่อกรณีที่ Alt คีย์ถูกกดร่วมด้วย

Previous Key State เป็นสถานะที่บอกให้ทราบว่า "ก่อนหน้านี้แป้นพิมพ์มีสถานะอย่างไร" ซึ่งจะเป็น 0 เมื่อก่อนหน้านี้คีย์ถูกปล่อยและเป็น 1 เมื่อก่อนหน้านี้คีย์ถูกกดอยู่

Transition State เป็น 0 เมื่อคีย์ถูกกด และ 1 เมื่อคีย์ถูกปล่อย ซึ่งบอกถึงสถานะปัจจุบันของคีย์บอร์ดในขณะที่เกิดเมสเสจขึ้นมา

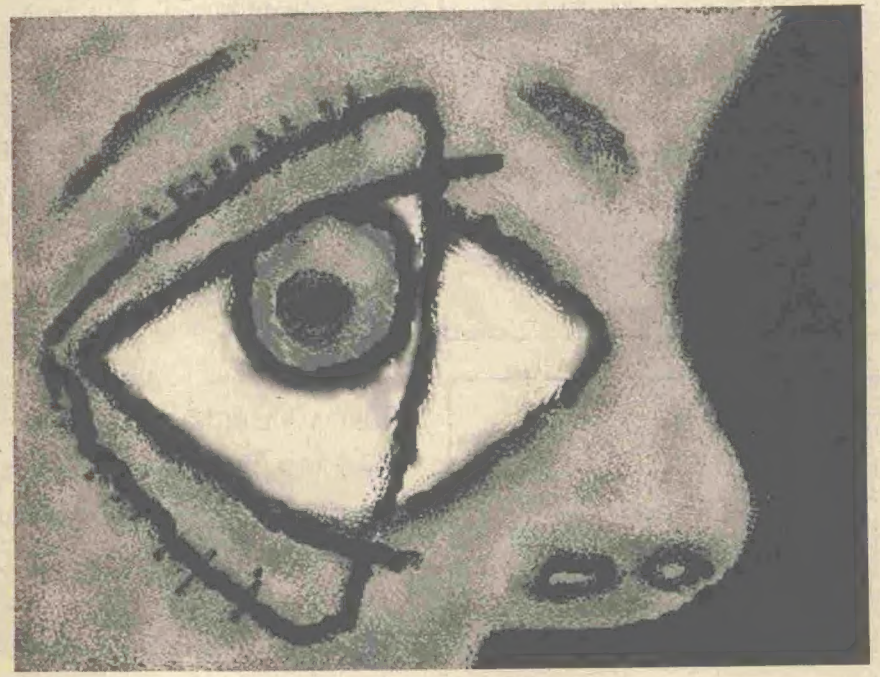
สำหรับการเขียนโปรแกรมจัดการกับเมสเสจที่กล่าวมาแล้วก็คงเป็นไปโดยที่แทรกเคส (case) ที่จะเช็คเข้าไปแล้วก็จัดการในส่วนโปรแกรมเอง แทนที่จะส่งผ่านให้ฟังก์ชัน DefWindowProc จัดการแทนเรา ดังตัวอย่างรูปที่ 3

จากตัวอย่างเราจะจัดการ WM_KEYDOWN แทนที่จะส่งผ่านให้ DefWindowProc ทำงานแทน

```
WindowProc (HWND hWnd, WORD wMess, WORD wParam, LONG lParam)
{
   switch (wMess)
      {
      case WM_CREATE:    ...
                         break;
      case WM_DESTROY:   ...
                         break;
      case WM_KEYDOWN:
                         if (wParam = = VK_NEXT) ...
                         ...
                         break;
      default: return (DefWindowProc (hWnd. wMess, wParam, lParam));
      }
return 0;
}
```

รูปที่ 3 ตัวอย่างการเขียนโปรแกรมจัดการกับเมสเสจ

ย้อนกลับไปดูรูปที่ 1 เราจะเห็นได้ว่า รหัสเทียมสำหรับตัวอักษรจะไม่มีการแยกแยะว่าเป็นตัวอักษรแบบใด ดังนั้นหากต้องแยกแยะว่าในขณะที่กดแป้นตัวอักษรนั้นเป็นตัวอักษรแบบใด ตัวพิมพ์ใหญ่หรือตัวพิมพ์เล็ก ก็คงต้องเป็นหน้าที่ของโปรแกรมเมอร์ที่ต้องเช็คเอง



คาแรกเตอร์เมสเสจ

จากที่เคยกล่าวมาแล้วเมื่อคุณกดหรือปล่อยแป้น จะมีคีย์สโตรคเมสเสจ แต่การจัดการ การเช็คเกี่ยวกับค่าตัวอักษรที่ได้ว่ามีสถานะอย่างไร เป็นเรื่องที่โปรแกรมเมอร์ต้องทำเอง เช่น เมื่อกดแป้นตัวอักษร A เมื่อ Caps Lock ON ค่าของ wParam ใน WM_KEYDOWN จะบอกเพียงว่าเป็นค่ารหัสเทียม A แต่จะไม่บอกว่า เป็นตัวพิมพ์ใหญ่ ซึ่งถ้าต้องการทราบเราจำเป็นที่จะต้องเช็คสถานะของแป้น Caps Lock ด้วยเราจึงจะทราบ ซึ่งดูไปแล้วคงต้องเขียนโค้ดในส่วนนี้เพิ่มขึ้นมากมาย

แต่จะสะดวกกว่าถ้าสามารถที่จะแปลงคีย์สโตรคเมสเสจนั้น ให้แสดงค่าในรูปเมสเสจที่ให้ค่าของแอสกีที่นำมาใช้ได้เลย โดยที่เราไม่ต้องจัดการอะไรมากมาย ซึ่งทางวินโดวส์เองเรียกเมสเสจแบบนี้ว่า คาแรกเตอร์เมสเสจ (Character Message)

การสร้างคาแรกเตอร์เมสเสจทำได้โดยเรียกผ่านฟังก์ชัน TranslateMessage

ใน GetMessage Loop

```
while (GetMessage (&msg, NULL, 0))
   {
   TranslateMessage (&msg);
   DispatchMessage (&msg);
   }
```

จากข้างบนเป็นส่วนของ GetMessage Loop ซึ่งเป็นพื้นฐานที่สุดที่ปรากฏใน WinMain โดยปกติแล้วฟังก์ชัน GEtMessage มีหน้าที่ดึงเอาเมสเสจออกจากคิวและเติมลงในโครงสร้าง MSG เพื่อนำไปใช้ต่อ และสำหรับฟังก์ชัน DispatchMessage จะเป็นส่วนที่นำข้อมูลที่ใส่ในโครงสร้าง MSG ไปรันส่วน WinProc

และตัวสำคัญคือ ฟังก์ชัน TranslateMessage ซึ่งทำหน้าที่แปลงคีย์สโตรคเมสเสจมาเป็นคาแรกเตอร์เมสเสจและส่งเข้าคิวตามหลังคีย์สโตรคเมสเสจนั้น ดังรูปที่ 4

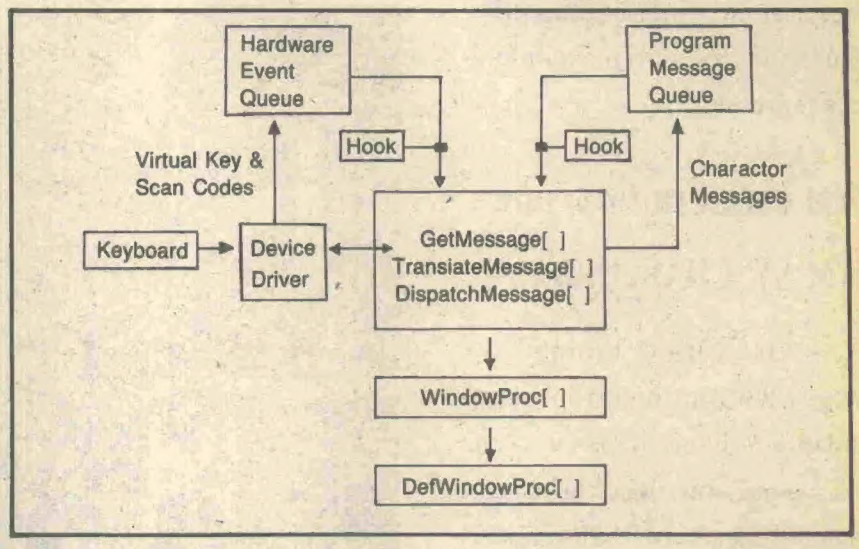
เมื่อกดแป้นตัวอักษร a หรือปล่อยแป้นตัวอักษร วงจรของเมสเสจที่เกิดขึ้นเมื่ออาศัยฟังก์ชัน TranslateMessage จะเป็นดังตารางที่ 1

กรณีที่ไม่เรียกผ่านฟังก์ชัน TranslateMessage วงจรของเมสเสจที่เกิดขึ้นจะเป็นดังตารางที่ 2

อีกตัวอย่างหนึ่ง เมื่อเรากดและปล่อยแป้นตัวอักษร W ควบคู่กับแป้น Shift วงจรของเมสเสจที่เกิดขึ้นจะได้ดังตารางที่ 3

สำหรับคาแรกเตอร์เมสเสจที่ได้จากฟังก์ชัน TranslateMessage แบ่งได้ตามชนิดคีย์สโตรคเมสเสจที่มาเป็น 2 กลุ่ม (รวม 4 เมสเสจ) ดังตารางที่ 4

คาแรกเตอร์เมสเสจที่อยู่ในกลุ่มนอนซิสเต็มส์คาแรกเตอร์ (Nonsystem Character) เกิดจากนอนซิสเต็มส์คีย์สโตรคเมสเสจ (Nonsystem Keystroke



รูปที่ 4 ฟังก์ชัน TranslateMessage

Message	Key or Code (wParam)
WM_KEYDOWN	Virtual Key A
WM_CHAR	ASCII Code a
WM_KEYUP	Virtual Key A

ตารางที่ 1 แสดงฟังก์ชัน TranslateMessage ในกรณีที่กดและปล่อยแป้นตัวอักษร a

Message	Key or Code (wParam)
WM_KEYDOWN	Virtual Key A
WM_KEYUP	Virtual Key A

ตารางที่ 2 แสดงฟังก์ชัน TranslateMessage ในกรณีไม่ผ่านฟังก์ชัน

Message	Key or Code (wParam)
WM_KEYDOWN	Virtual Key VK_SHIFT
WM_KEYDOWN	Virtual Key W
WM_CHAR	ASCII Code W
WM_KEYUP	Virtual Key W
WM_KEYUP	Virtual Key VK_SHIFT

ตารางที่ 3 แสดงฟังก์ชัน TranslateMessage ในกรณีที่กดและปล่อยแป้นตัวอักษร W

Type	Character	Dead Character
Nonsystem Character	WM_CHAR	WM_DEADCHAR
System Character	WM_SYSCHAR	WM_SYSDEADCHAR

ตารางที่ 4 แสดงชนิดของคีย์สโตรคเมสเสจ

Message) และสำหรับที่อยู่ในกลุ่มซิสเต็มส์คาแรกเตอร์เกิดจากซิสเต็มส์คีย์สโตรคเมสเสจ

WM CHAR เมสเสจ และ WM SYSCHAR เมสเสจ

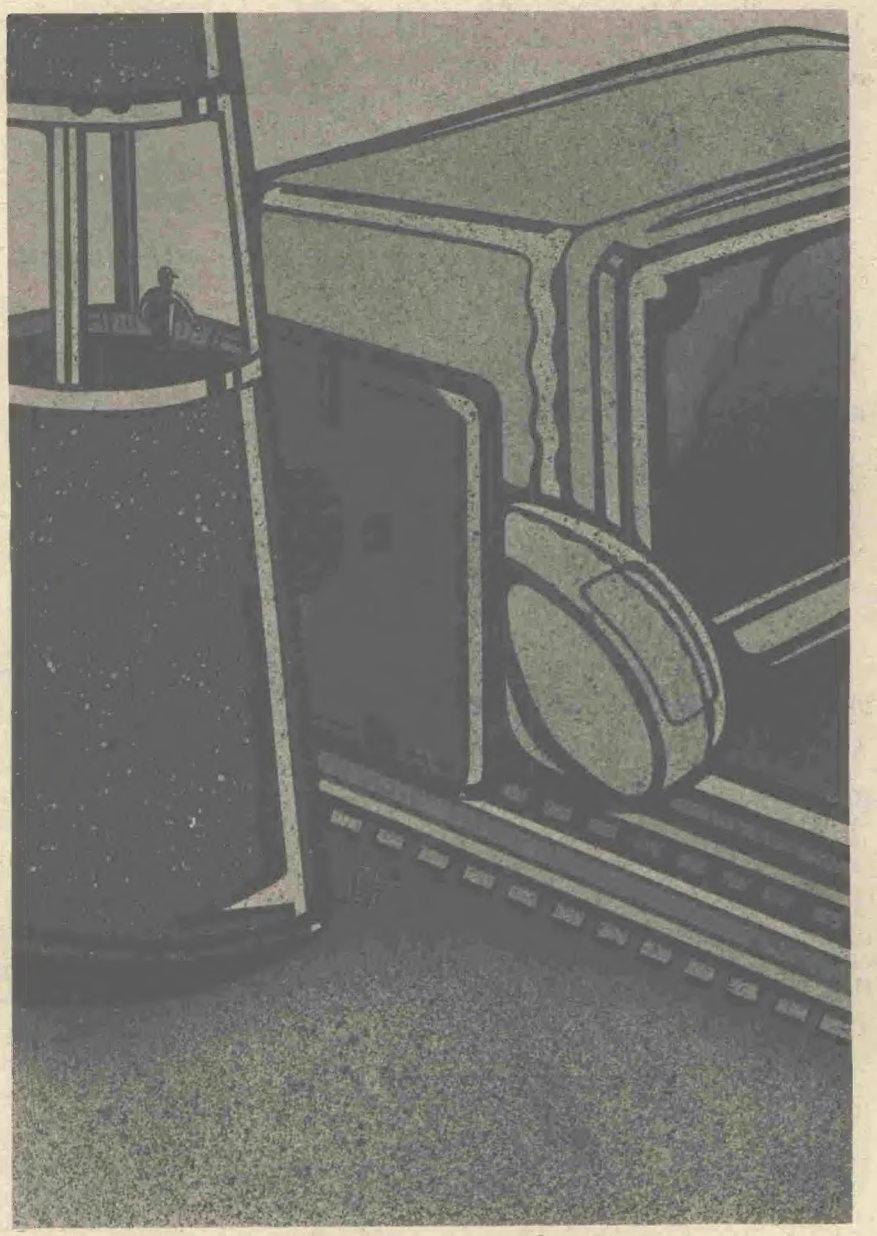
WM_CHAR เมสเสจ และ WM_SYSCHAR เมสเสจคือ คาแรกเตอร์เมสเสจที่เกิดจากฟังก์ชัน TranslateMessage ตามชนิดคีย์สโตรคเมสเสจ ซึ่งค่า wParam จะให้ค่าแอสกีที่เป็นจริงในขณะที่กดแป้นตัวอักษร และเราสามารถนำมาใช้ได้ทันที (เหมือนค่าแอสกีในรีจิสเตอร์ AL ที่ได้เมื่อเราเรียกผ่านฟังก์ชันของอินเตอร์รัพต์ 16h) รูปแบบการใช้มีดังรูปที่ 5

DEAD-CHARACTER

สำหรับคีย์บอร์ดโค้ดที่ไม่ใช่โค้ด U.S. เช่น โค้ด French หรือ โค้ด German จะมีตัวอักษรพิเศษ แต่ถึงจะเป็นคนละโค้ดแต่ก็เป็นคีย์บอร์ดชนิดเดียวกัน ดังนั้นอาศัยแป้นพิมพ์ที่ใช้ร่วมกันเพื่อสร้างโค้ดใหม่ ซึ่งแป้นพิมพ์ที่มีหน้าที่เช่นนี้เราเรียกว่า Dead Key โดยตัวมันแล้วไม่สามารถจะใช้โค้ดใด ๆ ได้ ต้องใช้สร้างโค้ดร่วมกับแป้นตัวอักษรอื่น

Keyboard Sample Program

สำหรับโปรแกรมตัวอย่างแสดงดังรูปที่ 6 จะเป็นโปรแกรมรายงานเกี่ยวกับเมสเสจที่เกิดขึ้น เมื่อมีการกดหรือปล่อยแป้นตัวอักษร เพื่อเป็นการยืนยันเนื้อหาข้างต้นที่คุยมาทั้งหมด



```
case WM_CHAR:
    switch (wParam)
    {
    case '\t' : ...      /* if key is Tab then done.          */
                break;
    case '\b': ...       /* if key is Backspace then done. */
                break;
    case 'a'  : ...      /* if key is a then done.            */
                break;
    }
    break;
```

รูปที่ 5 รูปแบบการใช้ฟังก์ชัน

รูปที่ 6 แสดงโปรแกรมตัวอย่าง

Keybrd.mak

```
#####################################################################
# Keybrd.mak : Makefile for completely translate to .exe file.
#####################################################################

     all: keybrd.exe

     COMPL_OPTION = -c -AS -Gsw -Oas -Zpe
     LINK_LIB = slibcew libw

     keybrd.res: keybrd.rc keybrd.h
         rc -r keybrd.rc
     keybrd.obj: keybrd.c keybrd.h
         cl $(COMPL_OPTION) keybrd.c
     keybrd.exe: keybrd.obj keybrd.def keybrd.res
         link /NOD keybrd, , ,$(LINK_LIB), keybrd.def
         rc keybrd.res
```

Keybrd.h

```
  #ifndef _KEYBRD
       #define  _KEYBRD
       int PASCAL WinMain(HANDLE, HANDLE, LPSTR, int);
       BOOL InitApplication(HANDLE);
       BOOL InitInstance(HANDLE, int);
       long FAR PASCAL MainWndProc(HWND, unsigned, WORD, LONG);
       void Show(HWND, char *, int, WORD);
  #endif
```

Keybrd.c →

```
/*************************************************************************
     PROGRAM: keybrd.c
     PURPOSE: Keyboard Input template for Windows applications
     FUNCTIONS:
          WinMain() - calls initialization function, processes message loop
          InitApplication() - initializes window data and registers window
          InitInstance() - saves instance handle and creates main window
          MainWndProc() - processes messages
*************************************************************************/
     #include "windows.h"
     #include "keybrd.h"

     HANDLE hInst;
     RECT rect;
     short cxChar, cyChar;

     typedef  struct {
         char  pzVir_key[13];
         short sVal;
     }VIR_KEY vir_key[] = {
          "VK_BACK     ", 0x08, "VK_TAB      ", 0x09,
          "VK_CLEAR    ", 0x0C, "VK_RETURN   ", 0x0D,
          "VK_SHIFT    ", 0x10, "VK_CONTROL  ", 0x11,
          "VK_MENU     ", 0x12, "VK_PAUSE    ", 0x13,
          "VK_CAPITAL  ", 0x14, "VK_ESCAPE   ", 0x1B,
          "VK_SPACE    ", 0x20, "VK_PRIOR    ", 0x21,
          "VK_NEXT     ", 0x22, "VK_END      ", 0x23,
          "VK_HOME     ", 0x24, "VK_LEFT     ", 0x25,
          "VK_UP       ", 0x26, "VK_RIGHT    ", 0x27,
```

→

รูปที่ 6 (ต่อ)

```
"VK_DOWN     ", 0x28, "VK_SELECT   ", 0x29,
"VK_PRINT    ", 0x2A, "VK_EXECUTE  ", 0x2B,
"VK_SNAPSHOT ", 0x2C, "VK_INSERT   ", 0x2D,
"VK_DELETE   ", 0x2E, "VK_HELP     ", 0x2F,
"VK_0        ", 0x30, "VK_1        ", 0x31,
"VK_2        ", 0x32, "VK_3        ", 0x33,
"VK_4        ", 0x34, "VK_5        ", 0x35,
"VK_6        ", 0x36, "VK_7        ", 0x37,
"VK_8        ", 0x38, "VK_9        ", 0x39,
"VK_A        ", 0x41, "VK_B        ", 0x42,
"VK_C        ", 0x43, "VK_D        ", 0x44,
"VK_E        ", 0x45, "VK_F        ", 0x46,
"VK_G        ", 0x47, "VK_H        ", 0x48,
"VK_I        ", 0x49, "VK_J        ", 0x4A,
"VK_K        ", 0x4B, "VK_L        ", 0x4C,
"VK_M        ", 0x4D, "VK_N        ", 0x4E,
"VK_O        ", 0x4F, "VK_P        ", 0x50,
"VK_Q        ", 0x51, "VK_R        ", 0x52,
"VK_S        ", 0x53, "VK_T        ", 0x54,
"VK_U        ", 0x55, "VK_W        ", 0x56,
"VK_X        ", 0x57, "VK_Y        ", 0x58,
"VK_Z        ", 0x59, "VK_NUMPAD0  ", 0x60,
"VK_NUMPAD1  ", 0x61, "VK_NUMPAD2  ", 0x62,
"VK_NUMPAD3  ", 0x63, "VK_NUMPAD4  ", 0x64,
"VK_NUMPAD5  ", 0x65, "VK_NUMPAD6  ", 0x66,
"VK_NUMPAD7  ", 0x67, "VK_NUMPAD8  ", 0x68,
"VK_NUMPAD9  ", 0x69, "VK_MULTIPLY ", 0x6A,
"VK_ADD      ", 0x6B, "VK_SEPARATOR", 0x6C,
"VK_SUBTRACT ", 0x6D, "VK_DECIMAL  ", 0x6E,
"VK_DIVIDE   ", 0x6F, "VK_F1       ", 0x70,
"VK_F2       ", 0x71, "VK_F3       ", 0x72,
"VK_F4       ", 0x73, "VK_F5       ", 0x74,
"VK_F6       ", 0x75, "VK_F7       ", 0x76,
"VK_F8       ", 0x77, "VK_F9       ", 0x78,
"VK_F10      ", 0x79, "VK_F11      ", 0x7A,
"VK_F12      ", 0x7B, "VK_NUMLOCK  ", 0x90,
"            ", 0x00


/*************************************************************************
    FUNCTION: WinMain(HANDLE, HANDLE, LPSTR, int)
    PURPOSE: calls initialization function, processes message loop
*************************************************************************/

int PASCAL WinMain(hInstance, hPrevInstance, lpCmdLine, nCmdShow)
HANDLE hInstance;
HANDLE hPrevInstance;
LPSTR lpCmdLine;
int nCmdShow;
{
    MSG msg;

    if (!hPrevInstance)
        if (!InitApplication(hInstance)) return (FALSE);

    if (!InitInstance(hInstance, nCmdShow)) return (FALSE);

    while (GetMessage(&msg, NULL, NULL, NULL)) {
```

รูปที่ 6 (ต่อ)

```
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
        return (msg.wParam);

/*************************************************************************
    FUNCTION: InitApplication(HANDLE)
    PURPOSE: Initializes window data and registers window class
*************************************************************************/
BOOL InitApplication(hInstance)
HANDLE hInstance;
{
    WNDCLASS  wc;

    wc.style = CS_HREDRAW | CS_VREDRAW;
    wc.lpfnWndProc = MainWndProc;
    wc.cbClsExtra  = 0;
    wc.cbWndExtra  = 0;
    wc.hInstance   = hInstance;
    wc.hIcon       = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor     = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = GetStockObject(WHITE_BRUSH);
    wc.lpszMenuName  =  "KeyMenu";
    wc.lpszClassName =  "KeyWClass";
    return (RegisterClass(&wc));
}

/*************************************************************************
    FUNCTION:  InitInstance(HANDLE, int)
    PURPOSE:  Saves instance handle and creates main window
*************************************************************************/
BOOL InitInstance(hInstance, nCmdShow)
HANDLE          hInstance;
int             nCmdShow;
{
    HWND            hWnd;
    HDC             hDC;
    hInst = hInstance;
    hWnd = CreateWindow( "KeyWClass", "Keyboard Sample Application",
                         WS_OVERLAPPEDWINDOW,
                         CW_USEDEFAULT, CW_USEDEFAULT,
                         CW_USEDEFAULT, CW_USEDEFAULT,
                         NULL, NULL, hInstance, NULL );
    if (!hWnd) return (FALSE);
    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);  /* It directly send WM_PAINT to update window. */
    return (TRUE);
}

/*************************************************************************
    FUNCTION: MainWndProc(HWND, unsigned, WORD, LONG)
    PURPOSE:  Processes messages
    MESSAGES:
        WM_CREATE     - create window
        WM_SIZE       - set scroll window size
        WM_PAINT      - update window, draw objects
        WM_KEYDOWN    - key pressed
        WM_SYSKEYDOWN - system key pressed
```

ซอฟต์แวร์
วินโดวส์โปรแกรมมิง ตอนที่ 4
: คีย์บอร์ดอินพุต

รูปที่ 6 (ต่อ)

```
        WM_KEYUP        - key released
        WM_SYSKEYUP     - system key released
        WM_CHAR         - ASCII character received
        WM_DESTROY      - destroy window

    COMMENTS:

        This demonstrates how keyboard messages are received, and what the
        additional information is that comes with the message.

********************************************************************/
long FAR PASCAL MainWndProc(hWnd, message, wParam, lParam)
HWND hWnd;
unsigned message;
WORD wParam;
LONG lParam;
{
    static char szHead_1[]="Message          Virtual_Key_Value       Char";
    static char szHead_2[]="---------------- ----------------------- ----";

    HDC hDC;                        /* display-context variable     */
    PAINTSTRUCT ps;                 /* paint structure              */
    TEXTMETRIC tm;

    switch (message) {

        case WM_CREATE:
                hDC = GetDC(hWnd);
                SelectObject(hDC, GetStockObject(SYSTEM_FIXED_FONT));
                GetTextMetrics(hDC, &tm);
                cxChar = tm.tmAveCharWidth;
                cyChar = tm.tmHeight;
                rect.top = 4*cyChar;
                ReleaseDC(hWnd, hDC);
                break;

        case WM_SIZE:
                rect.right = LOWORD(lParam);
                rect.bottom = HIWORD(lParam);
                UpdateWindow(hWnd);
                break;

        case WM_PAINT:
                InvalidateRect(hWnd, NULL, TRUE);
                hDC = BeginPaint(hWnd, &ps);
                SelectObject(hDC, GetStockObject(SYSTEM_FIXED_FONT));
                TextOut(hDC, cxChar, cyChar , szHead_1,
                        lstrlen(szHead_1));

            TextOut(hDC, cxChar, cyChar*2, szHead_2,
                    lstrlen(szHead_2));
            EndPaint(hWnd, &ps);
            break;
        case WM_KEYDOWN:
                Show(hWnd, "WM_KEYDOWN    ", 0, wParam);
                break;
        case WM_KEYUP:
                Show(hWnd, "WM_KEYUP      ", 0, wParam);
                break;
        case WM_SYSKEYDOWN:
                Show(hWnd, "WM_SYSKEYDOWN ", 0, wParam);
                return (DefWindowProc(hWnd, message, wParam, lParam));
```

รูปที่ 6 (ต่อ)

```
    case WM_SYSKEYUP:
              Show(hWnd, "WM_SYSKEYUP  ", 0, wParam);
              return (DefWindowProc(hWnd, message, wParam, lParam));

    case WM_CHAR:
              Show(hWnd, "WM_CHAR      ",  1, wParam);
              break;

    case WM_DESTROY:
        PostQuitMessage(0);
        break;

    default: return (DefWindowProc(hWnd, message, wParam, lParam));
}
return (NULL);
void Show(hWnd, szMessage, iType, wParam)
HWND hWnd;
char *szMessage;
int iType;
WORD wParam;
{
    static char szFormat[] = "%-17s%-24s  %c";
    char  szBuff[80];
    char  szTemp[13];
    HDC   hDC;
    int   i;

    ScrollWindow(hWnd, 0, -cyChar, &rect, &rect);
    if(iType == 0)
    {
    for(i=0; vir_key[i].sVal;i++)
       if(vir_key[i].sVal == wParam) break;
    if(vir_key[i].sVal) lstrcpy(szTemp, vir_key[i].pzVir_key);
    else lstrcpy(szTemp, "          ");
    }
   else
     lstrcpy(szTemp, "          ");
   hDC = GetDC(hWnd);
   SelectObject(hDC, GetStockObject(SYSTEM_FIXED_FONT));
   TextOut(hDC, cxChar, rect.bottom-cyChar, szBuff,
          wsprintf(szBuff, szFormat, (LPSTR) szMessage, (LPSTR) szTemp,
                   ((iType)? wParam : ' ')));
   ReleaseDC(hWnd, hDC);
   ValidateRect(hWnd, NULL);
```

Keybrd.def

```
NAME          Keyboard
DESCRIPTION   'Keyboard Sample Application written by N. Vichara.'
EXETYPE       WINDOWS
STUB          'WINSTUB.EXE'
CODE          PRELOAD MOVEABLE
DATA          PRELOAD MOVEABLE MULTIPLE
HEAPSIZE      1024
STACKSIZE     8192
EXPORTS       MainWndProc
```

Keybrd.rc

```
/* Notthing for resource but reserved for windows 3.0
   Keybrd is written by N. Vichara                    */
```

สำหรับซอร์สโค้ดข้างต้น คิดว่าผู้อ่านคงจะคุ้นเคยกับฟังก์ชันในโปรแกรมอยู่บ้าง สำหรับที่สำคัญก็เห็นจะเป็นฟังก์ชัน Show ซึ่งใช้รายงานชนิดของแป้นที่กดออกมา ฟังก์ชัน Show มีพารามิเตอร์ 4 ชุด ดังรูปที่ 7

หลักการทำงานของฟังก์ชัน Show เริ่มต้นด้วย Scroll เนื้อที่ของ (Window Client Area) แล้วเช็คชนิดการรายงาน กรณีที่ต้องการรายงานของคีย์สโตรค-เมสเสจก็จะเข้าไปนำชื่อ รหัสเทียม ในตารางมาเก็บที่ szTemp

จากนั้นการทำการพิมพ์รายงานโดยใช้ TextOut ออกมาแต่ก่อนที่จะพิมพ์ก็เลือกฟอนต์ที่ความกว้างของตัวอักษรเท่ากันหมด (ซึ่งทางวินโดวส์ได้เตรียมเป็นออปเจ็กต์ในสต็อกให้แล้ว) ให้แก่ดีไวซ์คอนเท็กซ์ (Device Context) (DC) ก่อนดังนี้

```
SelectObject (hDC, GetStockObject (SYS-
TEM _FIXED _FONT));
```

เมื่อรายงานเสร็จก็สั่ง ValidateRect เพื่อไม่ให้เกิดส่วนอัพเดตของพื้นที่และเป็นอันเสร็จสรรพสำหรับฟังก์ชัน Show

สำหรับรายละเอียดเกี่ยวกับฟังก์ชันต่าง ๆ ที่ไม่พูดถึง คงเหมือนเดิมสามารถหาดูได้จาก SDK Reference Manual ทั้งของไมโครซอฟต์และบอร์แลนด์

หมายเหตุ

สำหรับ Keybrd.rc ไม่ใช้ก็ได้โดยตัดออกจาก Make file แต่จะถูกเตือนตอนรันว่าเป็นโปรแกรมรูปแบบเก่าที่ใช้ในวินโดวส์ 2.0 ในกรณีนี้คอมพายล์ Keybrd.rc ด้วยเพื่อหลีกคำเตือนดังกล่าว และเนื้อแท้ใจความแล้ว Keybrd.rc ไม่มีอะไรเลย

hWnd	:	The handle of Window
szMessage	:	ชื่อของคีย์บอร์ดเมสเสจที่เรียกฟังก์ชัน Show
iType	:	ชนิดของคีย์บอร์ดเมสเสจเพื่อแยกชนิดการรายงาน iType = 0 เมื่อต้องการรายงานของคีย์สโตรคเมสเสจ iType ! = 0 เมื่อต้องการรายงานของคาแรกเตอร์เมสเสจ
wParam	:	ค่า wParam ค่าเมสเสจ

รูปที่ 7
แสดงพารามิเตอร์ของฟังก์ชัน Show

บทส่งท้าย

สำหรับฉบับนี้ ตัวผมเองคงจบเรื่องเกี่ยวกับคีย์บอร์ดอินพุตเพียงเท่านี้ แต่อย่างไรก็ตาม เรื่องของคีย์บอร์ดอินพุตมิได้มีเพียงสังเขปเท่านี้ ยังมีอะไรอีกที่น่าสนใจ ซึ่งก็หวังว่าผู้อ่านจะติดตามศึกษาต่อจากหนังสือ Programming Windows 3.0 หรือ Guide to Programming หรือ Peter Norton's Window 3 Power Programming Techniques หรือจากหนังสืออื่น ๆ

ในฉบับหน้าเราจะมาดูในเรื่องของเม้าส์อินพุตและไทม์เมอร์อินพุต พร้อมทั้งโปรแกรมตัวอย่างง่าย ๆ